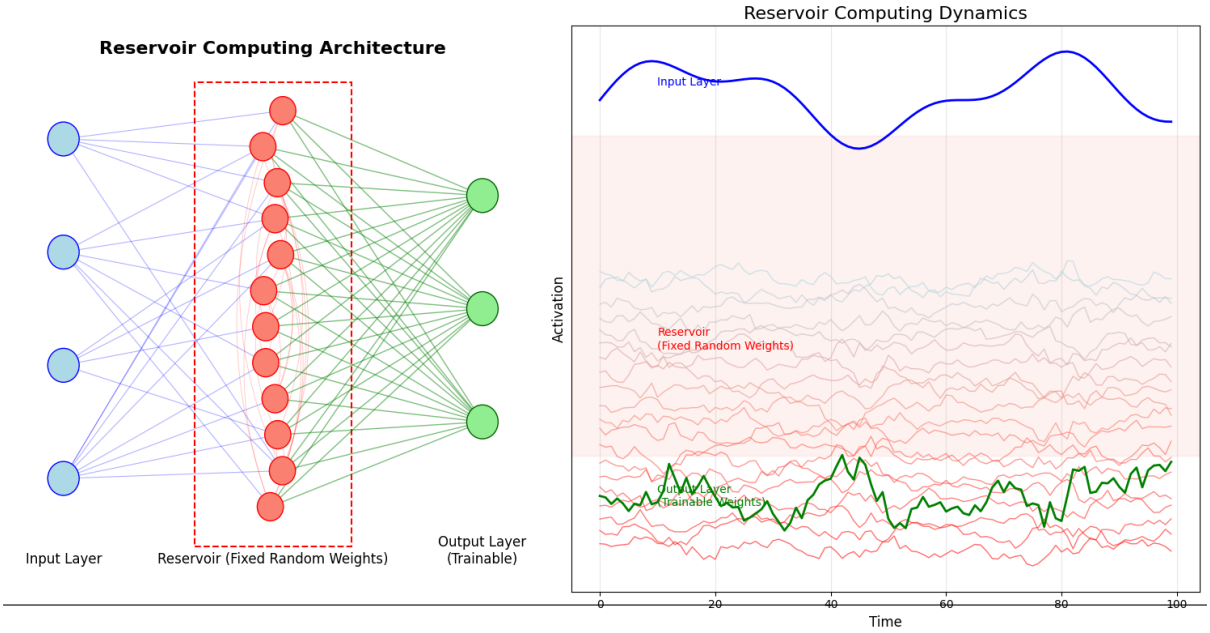


Reservoir Computing: Implementation and Analysis

Study Oriented Project

Vimarsh Shah

Final Report **April 2025**

BITS Pilani
K K Birla Goa Campus

Reservoir Computing: Implementation and Analysis

Vimarsh Shah
Department of Physics
BITS Pilani
Goa
f20221060@goa.bits-pilani.ac.in

April 2025

Abstract

Reservoir computing (RC) offers an alternative paradigm for analysing time series derived from dynamical systems. Utilising a fixed, randomly connected internal network (the reservoir), RC requires training only a linear output layer, thereby reducing the computational complexity associated with conventional recurrent neural network training. This report provides an overview of RC fundamentals, covering the echo state property necessary for processing temporal information, a comparison with standard recurrent networks, and a summary of its principal advantages and limitations.

The report also details implementations and applications of RC principles, specifically using an Extreme Learning Machine (ELM) as the time-series predictor, to analyse data from the logistic map. The input data consists of noisy time series generated at various parameter settings, designed to emulate data acquired from physical systems like electronic circuits subject to inherent noise. Principal Component Analysis (PCA) is applied to the output weights derived from training the ELM on these datasets, facilitating the estimation of the underlying parameter space (the bifurcation locus). Subsequently, the trained ELM is employed to generate the reconstructed bifurcation diagram across this estimated parameter range. We compare this architecture with other ones using different non-linear systems as well as Recurrent Neural Networks (RNNs). The results are presented, evaluating the capability of this RC-based approach to approximate the true system dynamics and its performance when dealing with the noise present in the input data.

The complete code with different tests and experiments conducted is available on GitHub.

Contents

1 Introduction

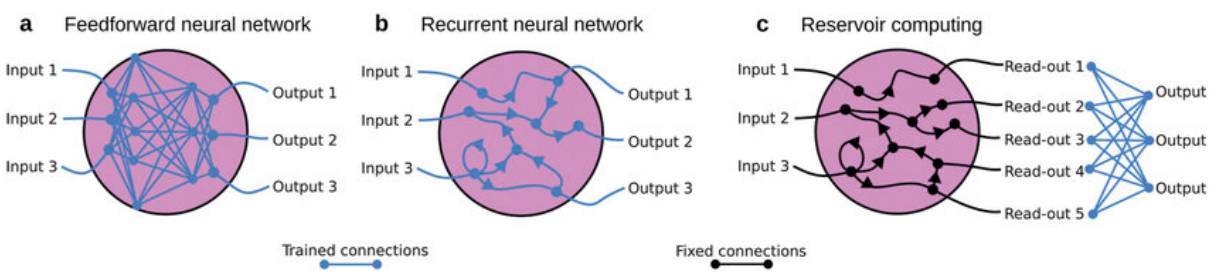
Machine learning has transformed numerous fields through its ability to extract patterns from data and make predictions without explicit programming. Traditional deep learning approaches often rely on gradient-based optimisation of neural network weights through backpropagation. While effective for many applications, these methods face challenges when dealing with recurrent neural networks (RNNs) designed to process sequential or temporal data.

RNNs are theoretically well-suited for temporal tasks due to their cyclical connections that enable information persistence across time steps. However, training RNNs presents significant difficulties, particularly the vanishing or exploding gradient problem during backpropagation through time as well the time taken to train a Neural Network. This challenge has motivated researchers to explore alternative approaches to harnessing the power of recurrent architectures.

Reservoir computing emerges in this context as a solution that maintains the representational power of recurrent networks while dramatically simplifying the training process. By leveraging a fixed, randomly initialised recurrent network (the reservoir) and only training the output connections, RC offers a computationally efficient approach to temporal modelling with several distinct advantages:

- Simplified training through linear regression of output weights only
- Reduced computational demands compared to full RNN training
- Ability to process temporal information through the reservoir's inherent memory
- Adaptability to physical implementations in various substrates

Figure 1: Illustration of (a) a feedforward neural network (FNN) with its acyclic and trainable connections, (b) a recurrent neural network featuring trainable feedback and cyclic connections, and (c) a reservoir made of untrained connections. For RC, only the connection between readout layer and output layer is trained, typically with a linear regression.



Cucchi et al. (2022)

This report explores the fundamentals of reservoir computing, examines implementation strategies, and analyzes results from reproducing experiments described in recent literature.

2 Basics of Reservoir Computing

2.1 General Principle

Reservoir computing is a computational framework that transforms input signals into output signals through a high-dimensional, nonlinear dynamical system (the reservoir). The key distinction from traditional neural networks is that the reservoir's internal connections remain fixed and random, while only the readout layer is trained. Cucchi et al. (2022)

The general workflow of reservoir computing consists of three main components:

- **Input Layer:** Maps external inputs to the reservoir through fixed, randomly initialised connection weights
- **Reservoir:** A recurrent network with random, fixed internal connections that creates a high-dimensional representation of the input
- **Output Layer:** A linear readout mechanism trained to map the reservoir states to desired outputs

When an input signal enters the system, it triggers complex dynamics within the reservoir, creating a high-dimensional nonlinear transformation of the original input. This transformation preserves temporal information from past inputs through the reservoir's recurrent connections. The output layer then extracts relevant features from this rich representation through a simple linear combination of reservoir states.

A typical discrete-time reservoir update can be written as

$$\mathbf{x}[n+1] = f\big(\mathbf{W}\mathbf{x}[n] + \mathbf{W}^{\text{in}}\mathbf{u}[n] + \mathbf{b}\big),$$

where $\mathbf{x}[n] \in \mathbb{R}^N$ is the reservoir state at time step n, $\mathbf{u}[n]$ is the input vector, $\mathbf{W}$ is the (fixed, random) reservoir weight matrix, $\mathbf{W}^{\text{in}}$ couples the input, and $f(\cdot)$ is a nonlinear activation (like *sigmoid* or tanh). The reservoir may also include a bias $\mathbf{b}$. The output of

the network is then formed as a linear combination of the reservoir states and possibly the inputs:

$$\mathbf{y}[n] = \mathbf{W}^{\text{out}}\big[\mathbf{x}[n];\, \mathbf{u}[n]\big],$$

where $\mathbf{W}^{\text{out}}$ is the output weight matrix to be trained. Only $\mathbf{W}^{\text{out}}$ is adjusted during training, typically by solving a linear regression (often ridge regression) to fit target output sequences. This greatly simplifies learning: by keeping $\mathbf{W}$ and $\mathbf{W}^{\text{in}}$ fixed and random, training reduces to a linear algebra problem. Cucchi et al. (2022) note that this means "one can use a random, untrained reservoir where only the output layer is optimised, for example, with linear regression". As a result, RC is extremely efficient to train compared to fully trainable recurrent networks, while still retaining rich nonlinear processing power.

While classical reservoirs are implemented in software simulations, an exciting trend is *physical reservoir computing*. Here, the reservoir is a physical system with inherent dynamics. For example, optical, mechanical, or spintronic devices with nonlinear responses and short-term memory can serve as reservoirs. Cucchi et al. (2022) emphasises that many material systems "show nonlinear behaviour and short-term memory that may be harnessed to design novel computational paradigms". Using a physical reservoir can drastically speed up computation and circumvent some implementation issues of digital neural networks. We will later discuss a specific example: using a single physical oscillator (a pendulum) as a reservoir Mandal et al. (2022).

2.2 Echo State Property and Fading Memory

A critical property of effective reservoirs is the Echo State Property (ESP), which ensures that the reservoir's state depends only on a history of inputs and not on its initial conditions. In practical terms, this means that any influence of the initial reservoir state eventually fades away, allowing the system to respond consistently to the same input sequence regardless of starting conditions.

This property creates what is known as *fading memory* - the reservoir retains information about recent inputs, with diminishing influence as inputs recede further into the past. This characteristic is essential for processing temporal information effectively.

Typically, this is achieved by scaling the spectral radius of $\mathbf{W}$ to be less than unity. In practice, a reservoir should have enough *memory capacity* to remember past inputs over a useful time horizon, but also enough nonlinearity to project inputs into a diverse feature space. The memory capacity quantifies how many past time steps the reservoir state

effectively retains. A larger reservoir (more neurons or nodes) generally increases capacity, but too much memory can harm generalisation Cucchi et al. (2022). Nonlinearity (like the tanh activation or other nonlinear dynamics) is crucial as well, because it allows the reservoir to create complex combinations of inputs; the output layer then linearly combines these to approximate arbitrary functions of the inputs.

2.3 Comparison with traditional RNNs

Unlike traditional RNNs where all connection weights are optimised through backpropagation, reservoir computing leaves the recurrent connections fixed, typically with random values. This approach circumvents the vanishing/exploding gradient problem and significantly reduces computational complexity.

The comparison can be summarised as follows:

Traditional RNNs: All connections (input, recurrent, and output) are trained through backpropagation through time, which is computationally intensive and may suffer from convergence issues.

Reservoir Computing: Only output connections are trained, typically through simple linear regression (ridge regression), while input and recurrent connections remain fixed.

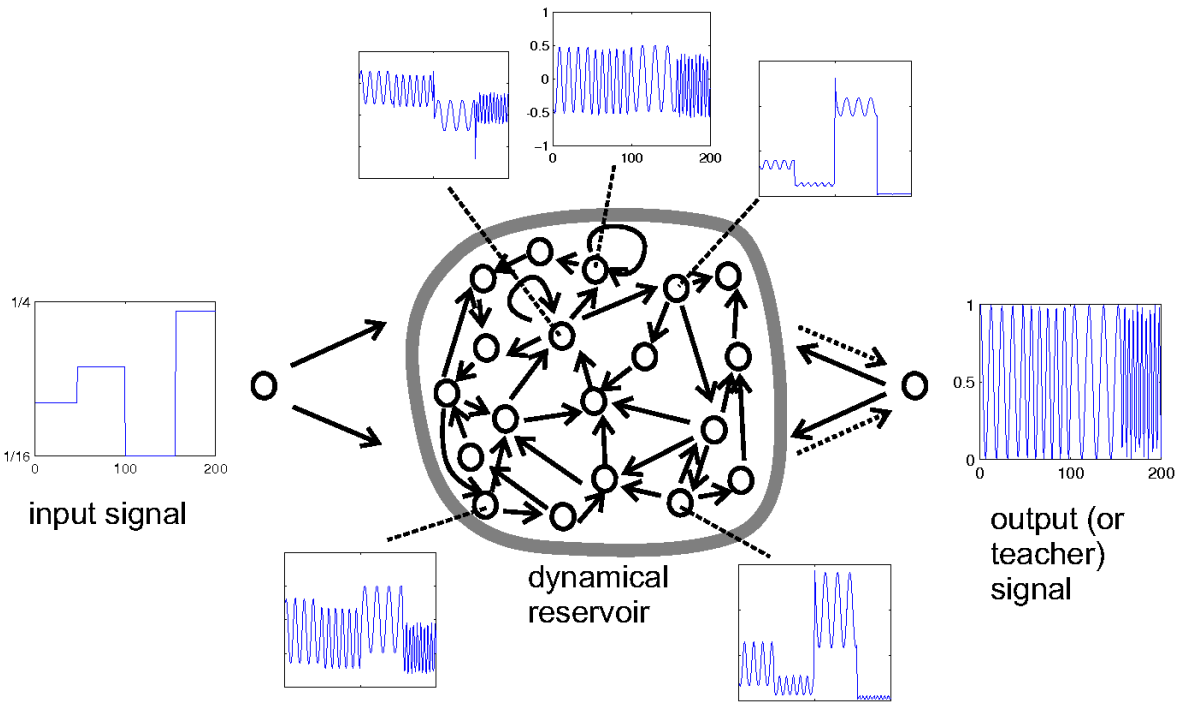
2.4 Some Varients

Two primary implementations of reservoir computing have been developed:

Echo State Networks (ESNs): Introduced by Jaeger (2001), ESNs typically use rate-based neurons with continuous activation functions like tanh, with sparse random connectivity

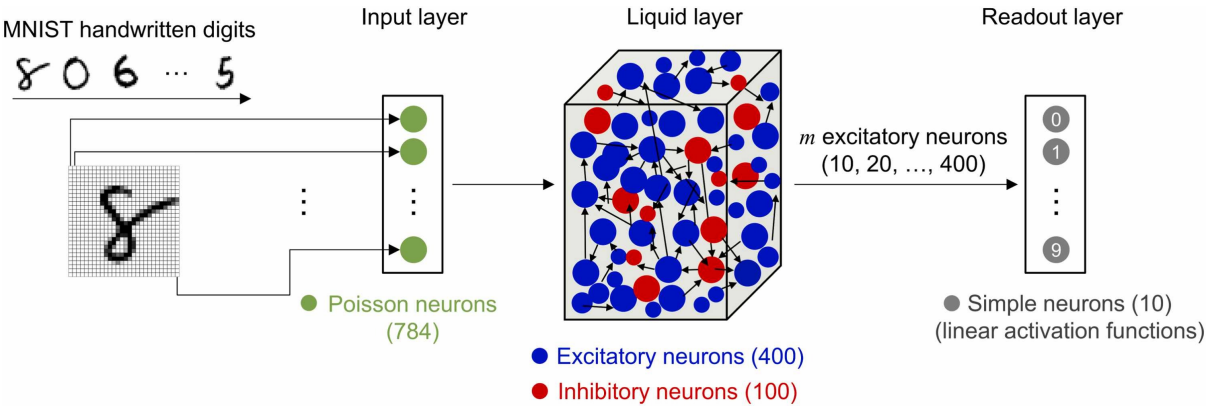
between reservoir units.

Figure 2: Echo State Network diagram. Commons (2024)



Liquid State Machines (LSMs): Developed by Maass (2010), LSMs use spiking neuron models that more closely resemble biological neurons. They were initially conceived in the context of computational neuroscience.

Figure 3: Liquid State Machine diagram. Woo et al. (2024)



Additionally, physical reservoir computing (PRC) extends these concepts by implementing

reservoirs in physical substrates rather than simulating them digitally. This allows exploitation of inherent physical dynamics in materials and systems for computation. As done in one of the papers I worked on implementing: Mandal et al. (2022).

2.5 Training Methodology

The training procedure for reservoir computing is remarkably straightforward compared to traditional neural networks:

- Feed input signals into the reservoir and collect the resulting reservoir states
- Train a linear readout function (through ridge regression) to map reservoir states to desired outputs
- Apply the trained readout to new reservoir states during inference

The mathematical formulation for this training is typically:

$$W_{\text{out}} = YG^T \left(GG^T + \lambda I\right)^{-1}$$

Where W_{out} represents the output weights, Y contains the target outputs, G contains the reservoir states, λ is a regularisation parameter, and I is the identity matrix.

2.6 Advantages and Limitations

Reservoir computing offers several distinct advantages:

- Training involves only linear regression, avoiding complex and computationally intensive backpropagation.
- The same reservoir can be used for multiple tasks by training different readout functions.
- The fixed nature of the reservoir enables implementation in physical systems.

However, there are important limitations as mentioned by Zhang and Cornelius (2023):

- The random initialisation provides little control over specific functionalities.
- The memory capacity of the reservoir is constrained by its size.
- Performance can vary significantly based on the specific task and how well it matches the reservoir's dynamics. Cucchi et al. (2022)
- The system in many testing is very much dependent on hyperparameters, and the type of non linear system you select for a particular task.

3 Literature Review

As seen in this exploratory article Cucchi et al. (2022), we see how to set up an RC experiment with a given nonlinear system, and highlight the advantage that only the output layer needs training.

Zhang and Cornelius (2023) explores limitations of standard RC models in what they call the "Catch-22s". They study the problem of basin prediction in multistable systems: determining from an initial condition which attractor the system will approach. They find that a classic echo state network requires a very long *warm-up* (nearly the full transient trajectory) to correctly predict the basin. In contrast, a variant called next-generation RC (NGRC) that encodes the exact nonlinear equations can predict basins with almost no warm-up. However, NGRC's performance is extremely sensitive: even tiny errors in the presumed nonlinearities make its predictions fail. In summary, this work shows that standard RC works well for some tasks but struggles with complex basin structure unless aided by precise system knowledge.

Arun et al. (2024) propose using a discrete nonlinear map as the reservoir. Specifically, they use the logistic map and a finite trigonometric series to create *virtual nodes* of a reservoir. They demonstrate that this simple reservoir can accurately predict several benchmark tasks: the Lorenz, Rössler, and Hindmarsh–Rose systems (temporal tasks) and even a seventh-order polynomial (non-temporal). Remarkably, their logistic-map reservoir works well even when noisy inputs are present. The authors report low prediction error (RMSE), indicating the logistic map can serve as an effective high-dimensional reservoir. Their approach "removes the necessity of continuous dynamical systems for constructing the reservoir" and suggests a general method for time-series prediction using simple discrete maps.

Mandal et al. (2022) examine a physical example of a minimal reservoir. They use a single driven pendulum (a simple nonlinear mechanical oscillator) as the reservoir. Through both simulation and a proof-of-principle experiment, they show that this one-dimensional system can perform standard RC tasks. The key is to exploit the pendulum's rich transient dynamics: they feed an input signal as a driving torque, record the pendulum's angle and angular velocity over time, and train the output on those signals. Surprisingly, even this "minimal one-node reservoir" achieved good accuracy on temporal (predicting time series) and non-temporal tasks. This work indicates that even a very low-dimensional physical system can have considerable "learning potential" when used as a reservoir.

Finally, Itoh et al. (2020) focus on reconstructing bifurcation diagrams from data. They generate time-series data from an electronic circuit (a discrete-time chaotic system)

across different parameter values, including dynamical and observational noise. Using an Extreme Learning Machine (ELM) – a single-hidden-layer feedforward network with random weights – they train a predictor to map time-series inputs to future values. By sweeping the input parameter and using the ELM predictions, they successfully reconstruct the system's bifurcation diagram. Importantly, their method proved robust: the reconstructed Lyapunov exponents matched those from the true system, showing accurate attractor structure recovery even in noisy conditions. This demonstrates that machine learning (via ELM) can infer global bifurcation behaviour from limited time-series data.

4 Implementation and Results

This section contains some of the experiments that I implemented and tested about various reservoir computing systems, inspired by the literature above.

4.1 Pendulum Reservoir

Firstly, I simulated a driven pendulum as a reservoir, following the approach of Mandal et al. (2022). The pendulum dynamics are given by the standard nonlinear equation (including damping and an external driving torque).

As the input signal, we inject a time-varying driving force as the reservoir state, and record the pendulum's angle and angular velocity. These two scalar values at each time were concatenated into a state vector. Over a sequence of inputs, the pendulum produces a rich time series in its transient motion.

The dynamics of the driven damped pendulum are governed by the following system of differential equations:

$$\begin{aligned} \frac{dx}{dt} &= v, \\ \frac{dv}{dt} &= -\frac{g}{l}\sin(x) - kv + f\,\text{sign}(\sin(\omega t)), \end{aligned} \tag{1}$$

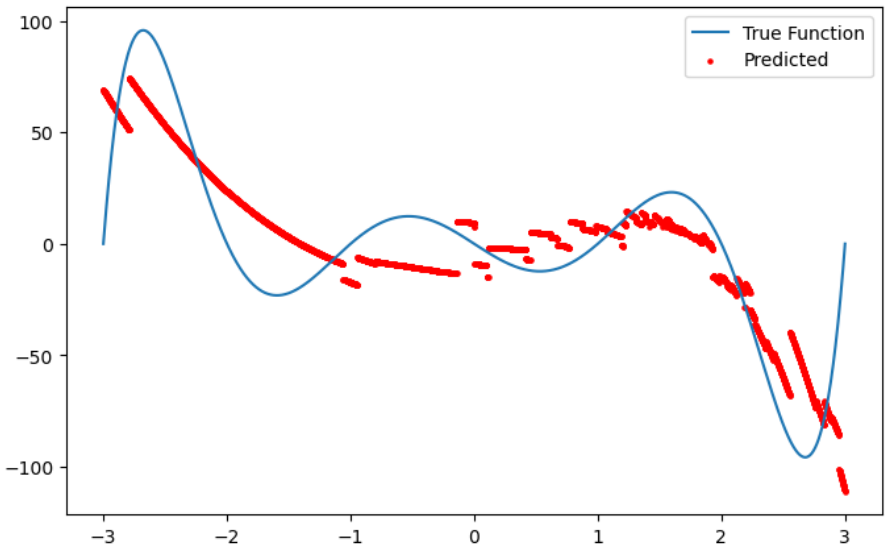
where

- $x(t)$ is the angular displacement,
- $v(t)$ is the angular velocity,
- g is the gravitational acceleration,
- l is the length of the pendulum,

- k is the damping coefficient,
- f is the amplitude of the external periodic forcing,
- ω is the forcing frequency,
- and sign$(\cdot)$ is the signum function.

We collected these state histories and trained a linear output layer to predict a target time series (for example, reconstructing the input signal after a delay). We found that the pendulum reservoir could learn the task to a useful degree of accuracy, similar to the findings of the paper. Figure 4 shows the results of a predicted graph just based on x_{t-1}. I believe I could get better results, and that it is not perfect due to RC being highly dependent on the hyperparameters set. But this experiment shows that a simple single oscillator can function as an effective reservoir for learning time-dependent tasks.

Figure 4: Results of the single pendulum reservoir experiment.



The Lorenz system is a classic example of a chaotic dynamical system, described by the following set of ordinary differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z, \end{aligned} \tag{2}$$

where σ, ρ, and β are positive parameters controlling the dynamics. In this standard

Lorenz system, the parameters are set to

$$\sigma = 10, \quad \rho = 28, \quad \beta = \frac{8}{3}.$$

These equations are solved numerically with an initial condition $\mathbf{x}(0) = (x_0, y_0, z_0)$. In this project, time series data is generated by integrating the Lorenz equations with initial state $(1, 1, 1)$ over a time span of $t \in [0, 100]$ using 10,000 points.

For training the reservoir computing model, the x-component of the Lorenz system is used as the input sequence, and the reservoir is trained to predict the full state vector (x, y, z) at the next time step.

Figure 5: Actual vs Predicted for the Lorentz system using Pendulum reservoir

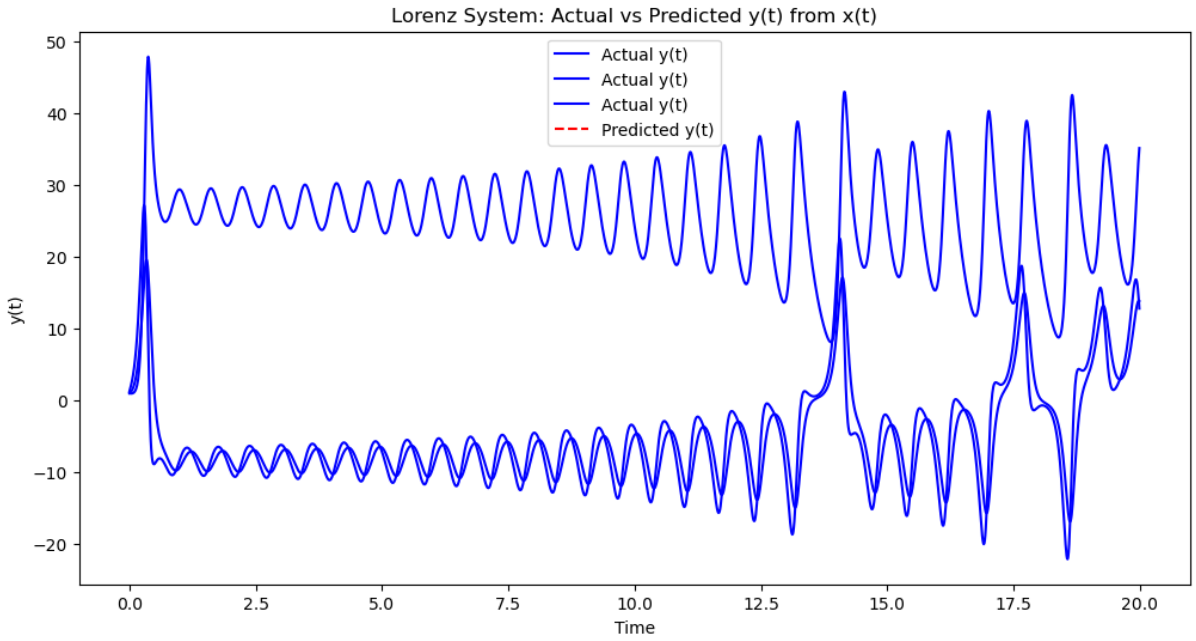
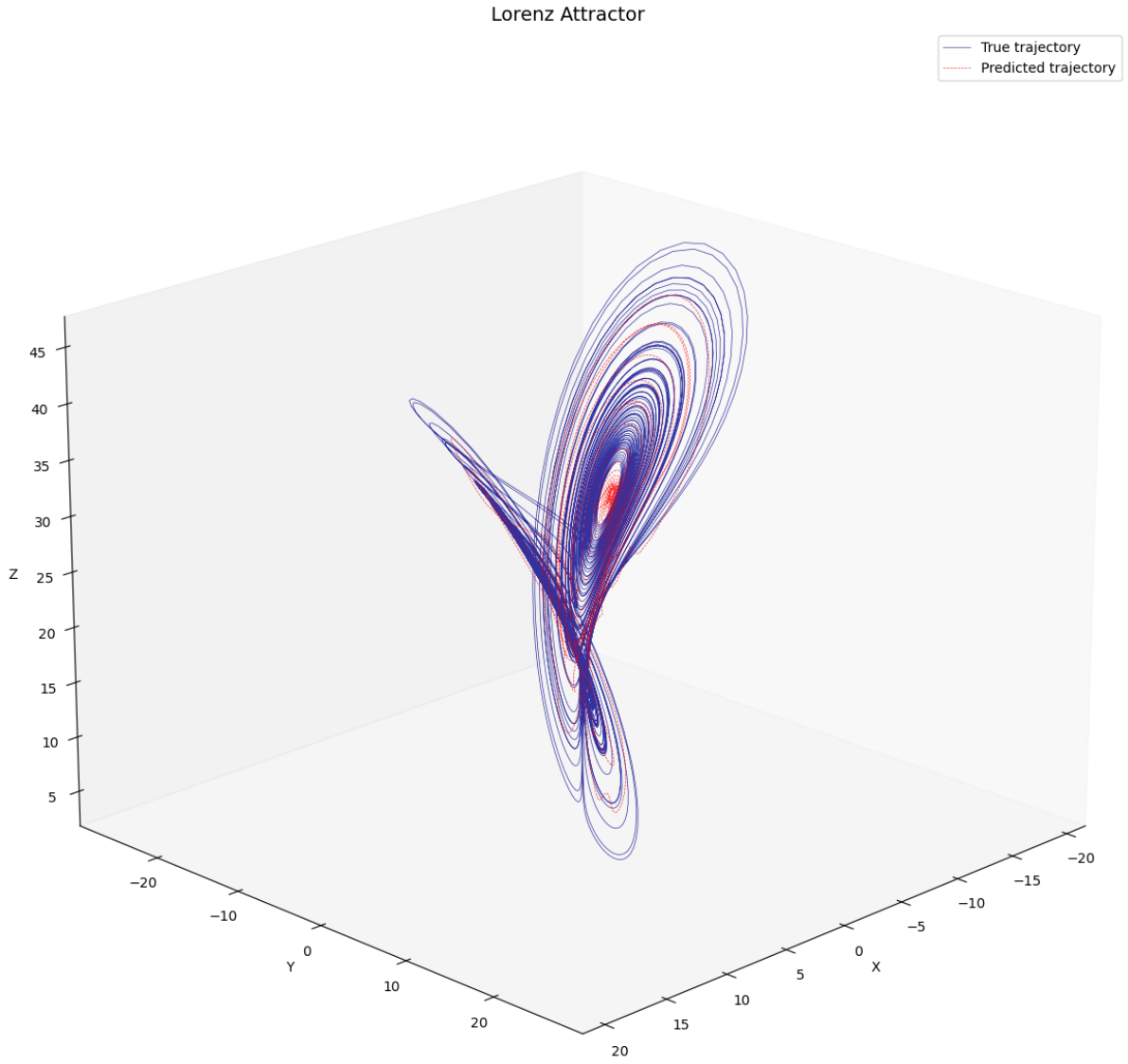


Figure 6: 3D render of actual vs Predicted timestep



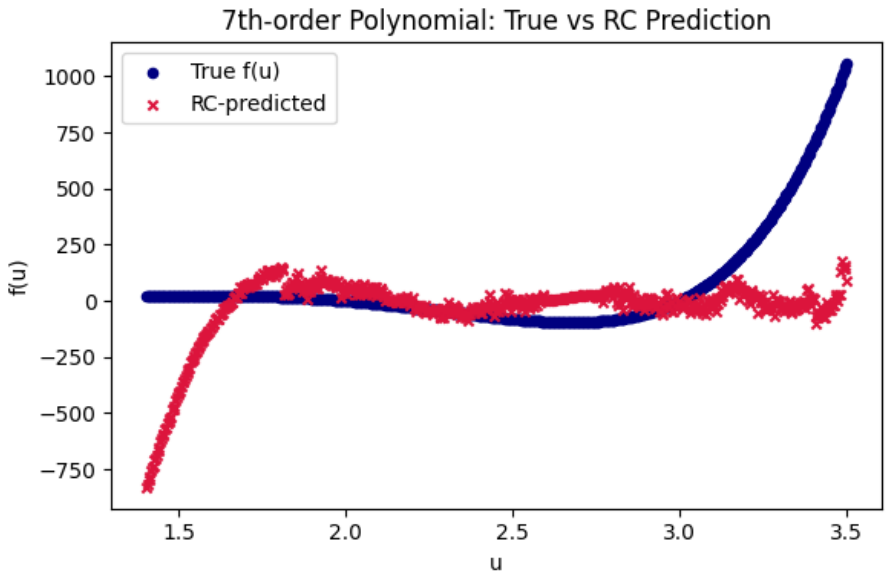
4.2 Logistic Map Reservoir

A reservoir was also constructed using the logistic map, as described by Arun et al. (2024). The logistic map $x_{n+1} = rx_n(1 - x_n)$ is iterated to generate a sequence. To create a high-dimensional reservoir, the technique of *virtual nodes* is used: for each new input, the

logistic map is updated several times (with fixed parameter r) and its values sampled at intermediate steps, possibly combined with a simple trigonometric expansion as in the paper. Here, a 7th-degree polynomial was set as a micro-benchmark, which is used in many other reservoir computing papers as well.

These sampled values form the reservoir state vector. The output layer is then trained on this state to predict a time-series target (for example, forecasting the future state of a chaotic system). The results (as seen in Figure 7) show that the logistic-map reservoir could indeed predict a chaotic time series with relatively low error. Noise perturbations in the input had only a minor effect, consistent with the robustness reported by the paper as well. This validates that even a discrete map can act as an effective reservoir model.

Figure 7: Results using Logistic Map to then try to map to a target polynomial.



Now, as implemented for Pendulum Reservoir, a similar problem is tried but using this logisitic map's system. Doing that the following results are obtained: 8 9.

Test RMSE (x,y,z): [3.76673884 2.93538559 8.36222649]

Figure 8: Results for the Lorenz attractor using Logistic Map

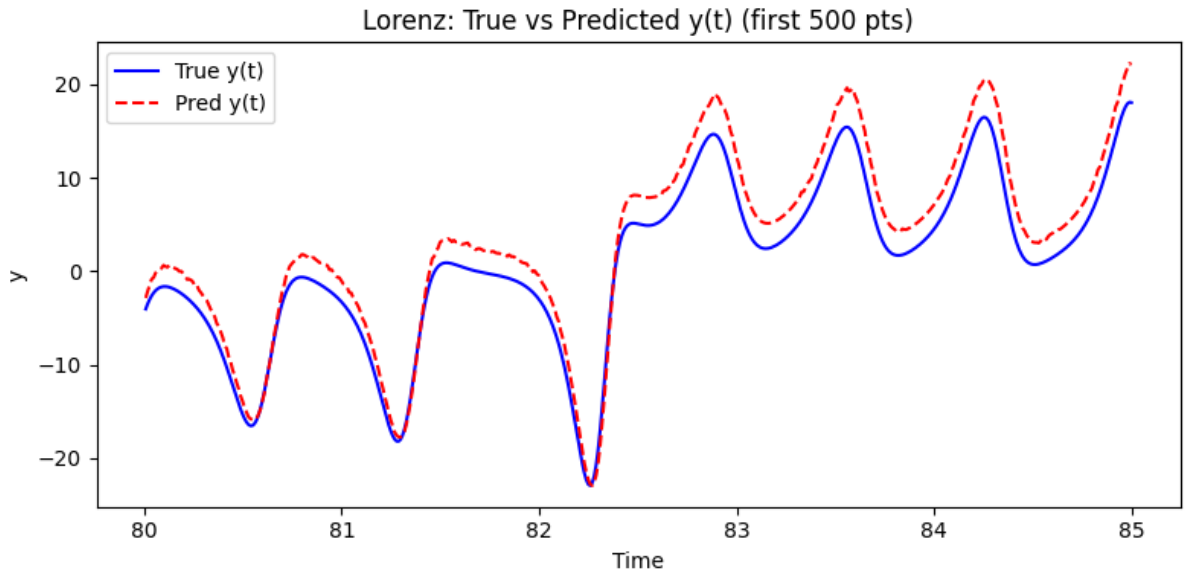
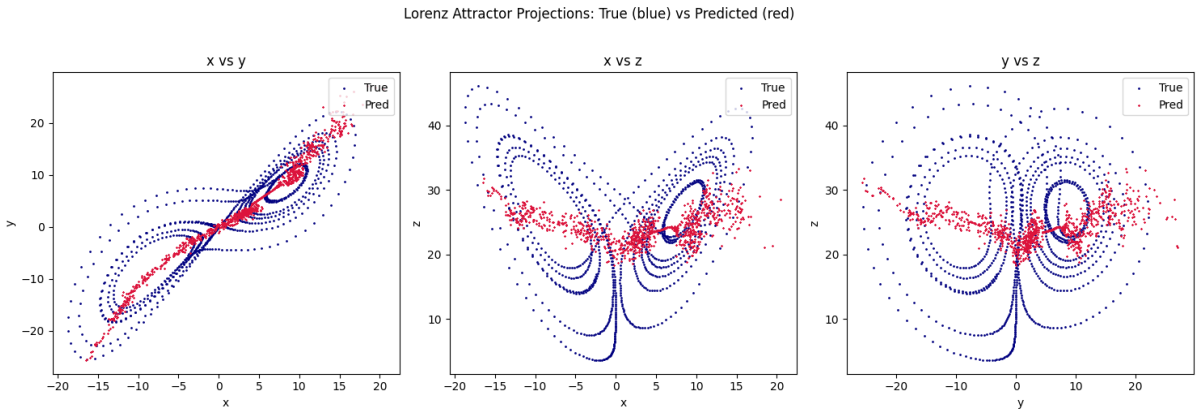


Figure 9: Results for the Lorenz attractor using Logistic Map, evolution of different axes over time.



4.3 Bifurcation Reconstruction via PCA and ELM

Reconstruction of bifurcation diagram of a nonlinear system using reservoir outputs, inspired by Itoh et al. (2020) is tried. The procedure is as follows: for a range of a system

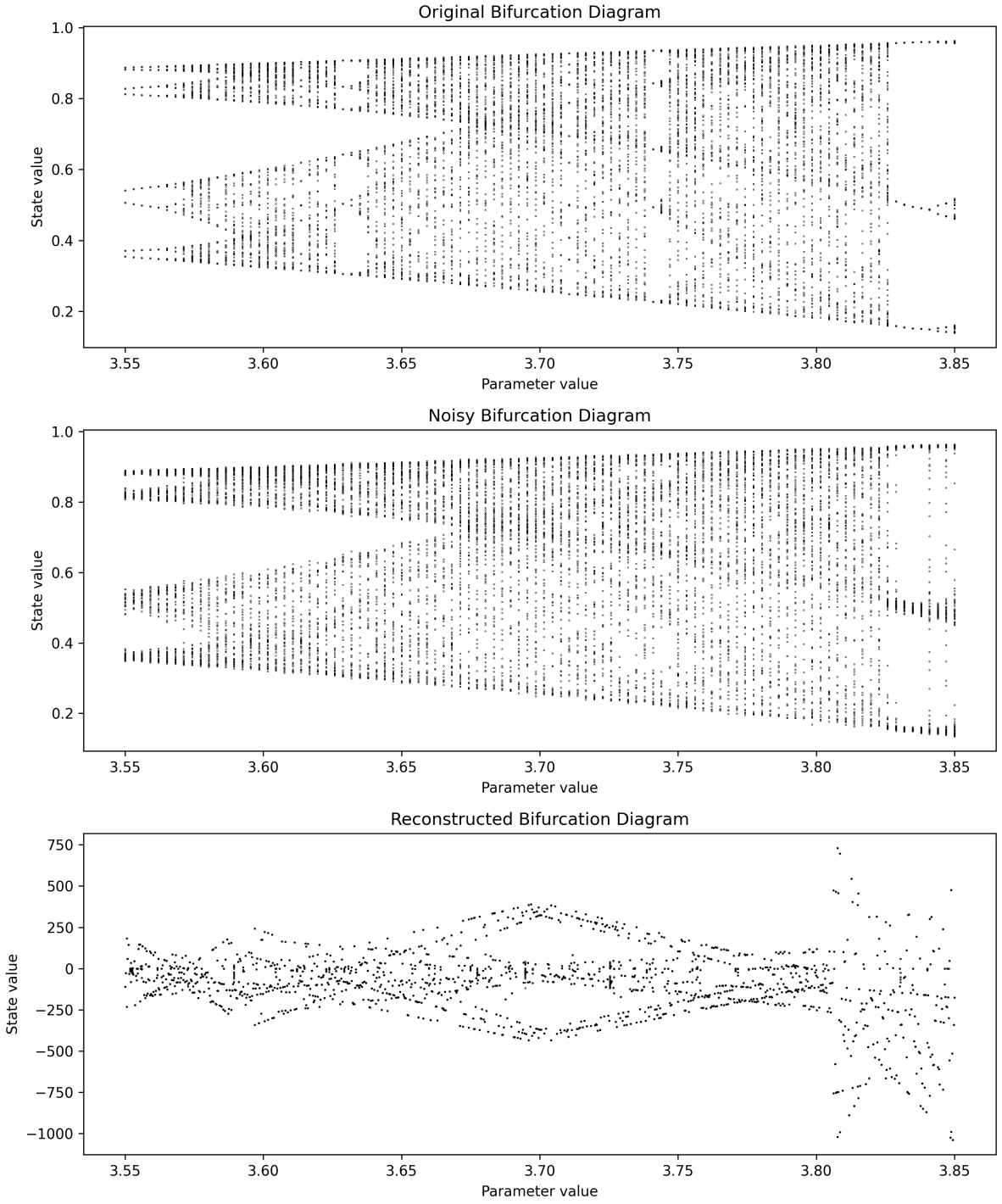
parameter μ, run the simulations of the system and feed its time-series data through the reservoir (logistic reservoirs).

The reservoir state responses are collected for each parameter value. Principal Component Analysis (PCA) is applied to reduce dimensionality to the reservoir states, extracting the leading modes as features. Then an Extreme Learning Machine (ELM) regressor is trained to map these PCA features (plus the parameter μ) to a predicted system output (the next state or an observable). By sweeping μ and using the trained ELM predictions, the goal is to trace the transitions of attractor values versus μ.

Figure 10 shows results for the reconstructed diagram. It is found that reconstruction is incomplete: some bifurcation branches are missing, distorted, or just weird. The ELM managed to learn local trends of the data to some extent, but it does not capture the full global structure. This was in contrast to the paper that achieved robust reconstructions. The imperfect result may be due to limited training data, noise, an insufficient reservoir or wrong implementation. The authors have not shared their implementation, just a vague description of hyperparameters. The replicated implementation could also have a high

chance of being incorrect/incomplete to an extent.

Figure 10: The first attempt at reconstruction of a system's bifurcation diagram using reservoir PCA + ELM



After some more fine-tuning and fixing a lot of issues, these are some results that seem

much better than the original ones.

Figure 11: Training MSE: 1.06e-05. Code for this implementation:

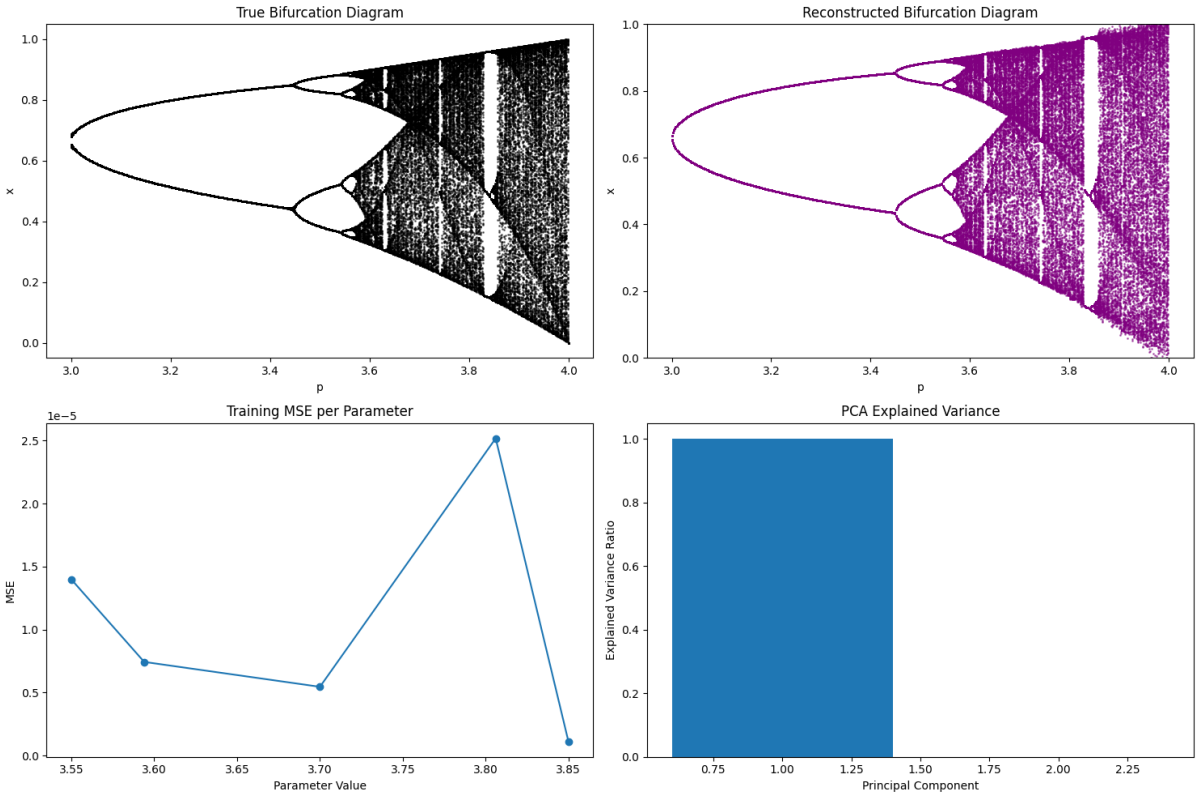


Figure 12: One of the visualisations in the paper was reconstructing the return plot, i.e., $x_{(t+1)}$ vs x_t

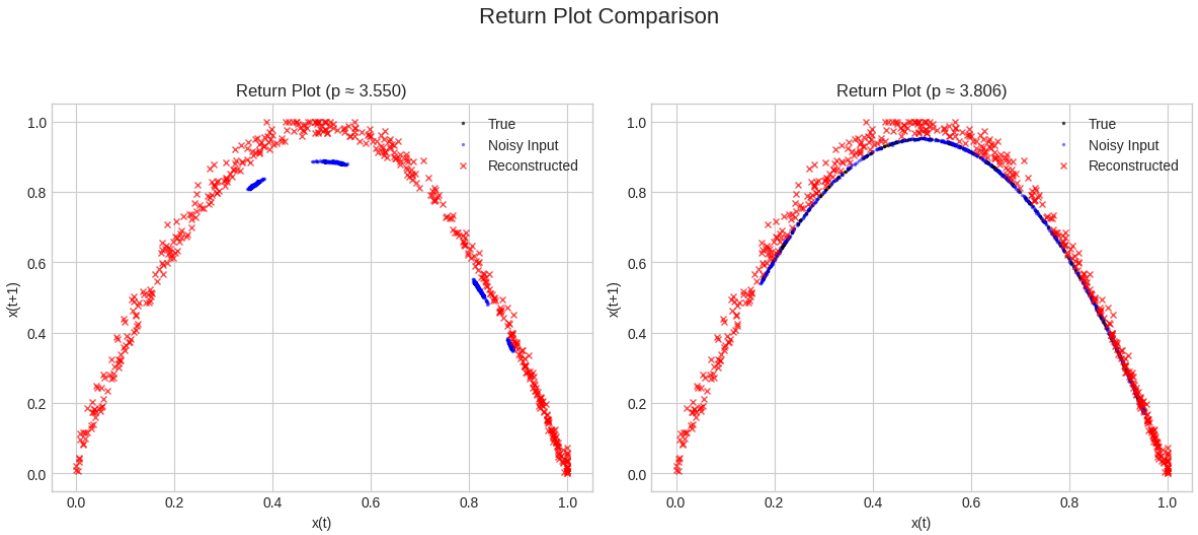


Figure 13: Plot of the overlapping true and predicted bifurcation diagrams. The issue remains that the predictions outside the trained parameter space are not very accurate.

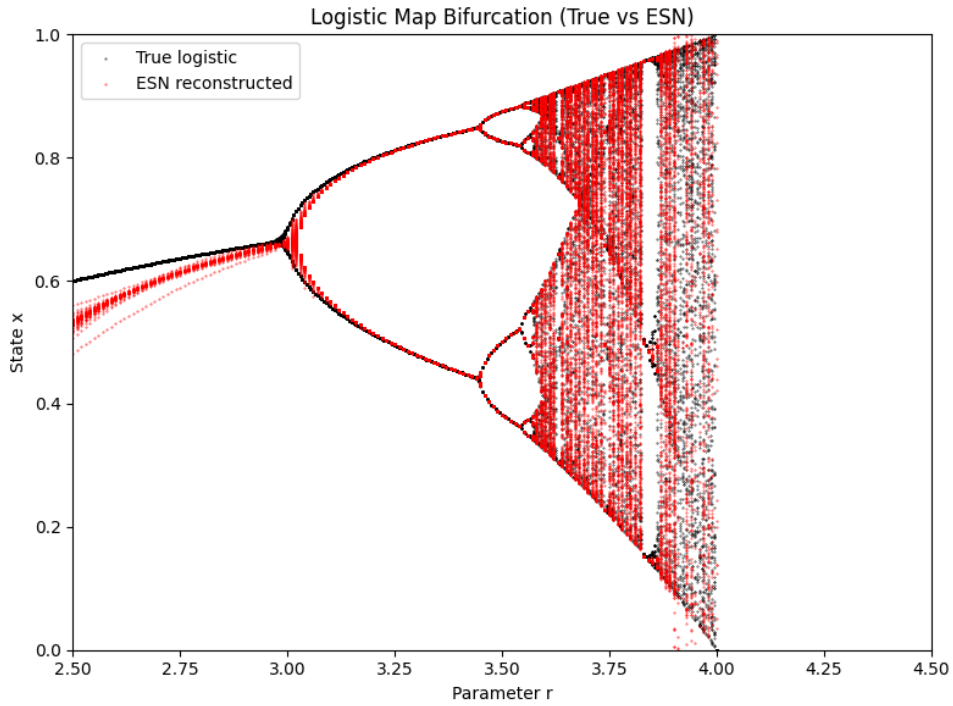


Figure 14: Using the reservoir model to estimate approximate Lyapunov exponents for different parameter values. The system as only trained on the highlighted green region of the graphs, it is estimating the exponent for other parameter values.

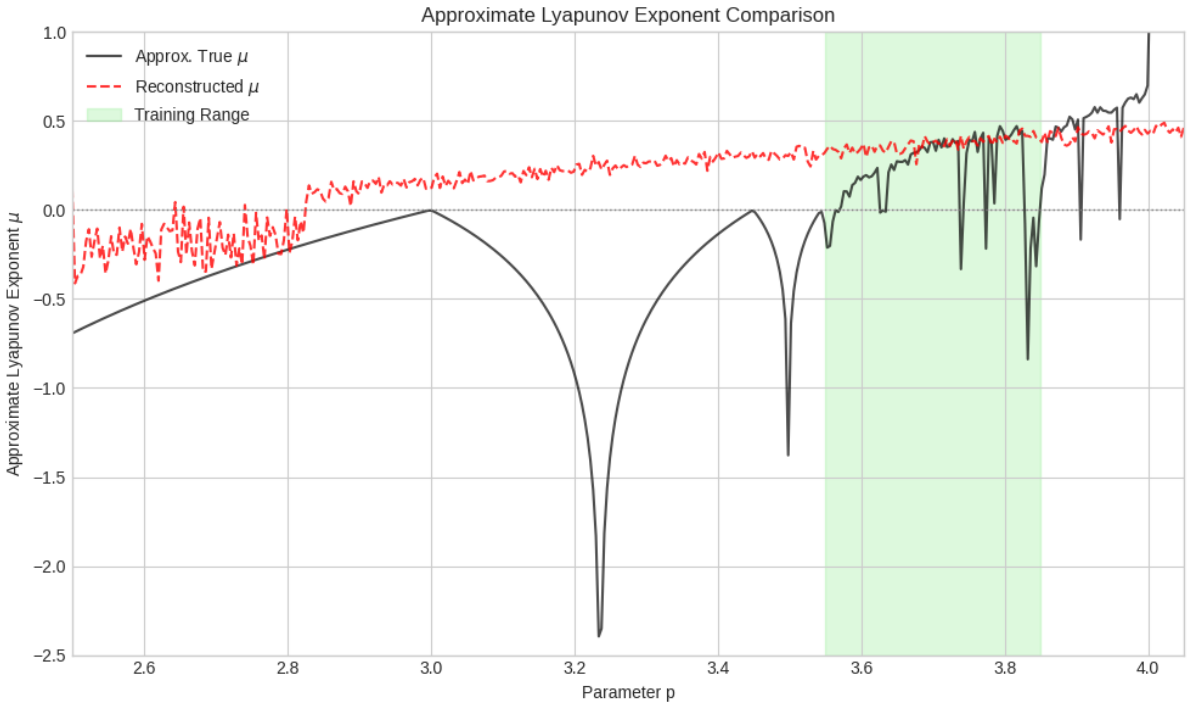


Figure 15: Analysis of the PCA components for different parameters yields an interesting correlation. Almost like a straight line being fit through the values with slope of -1.

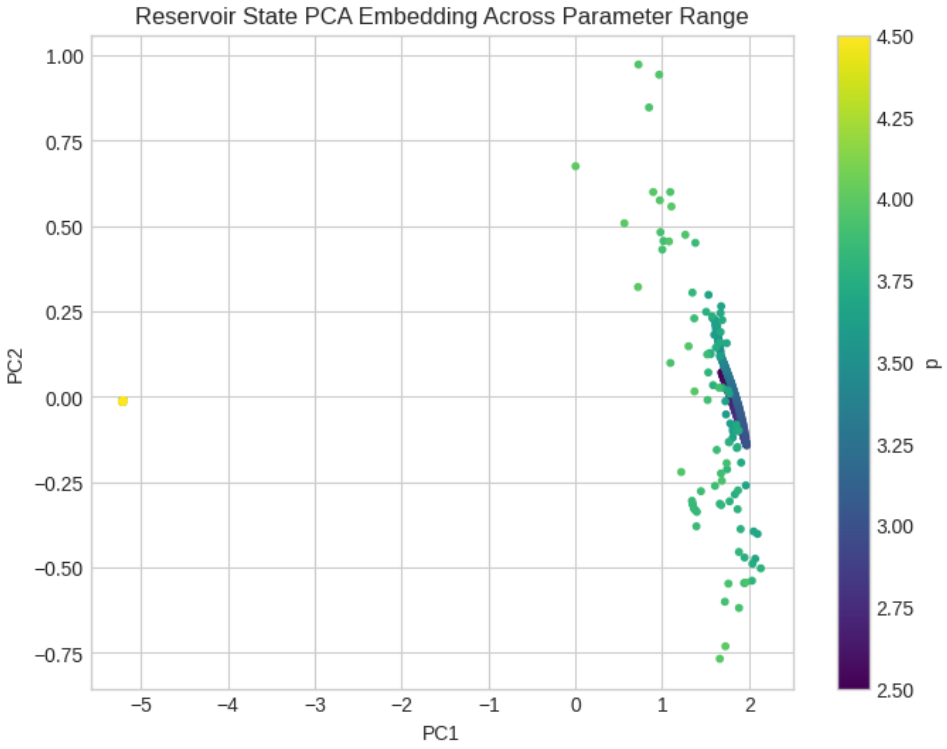
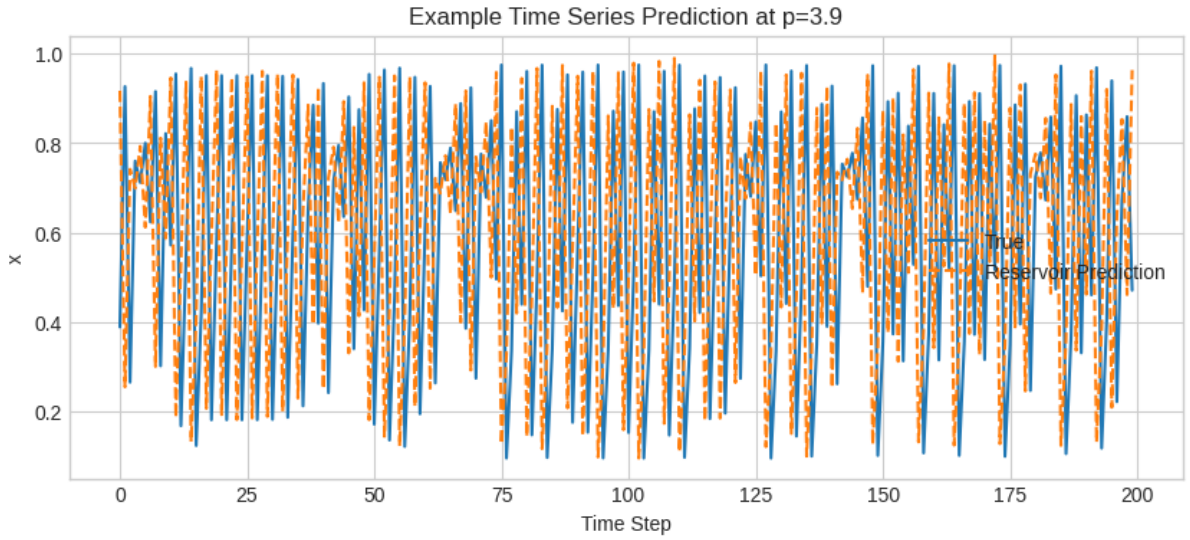


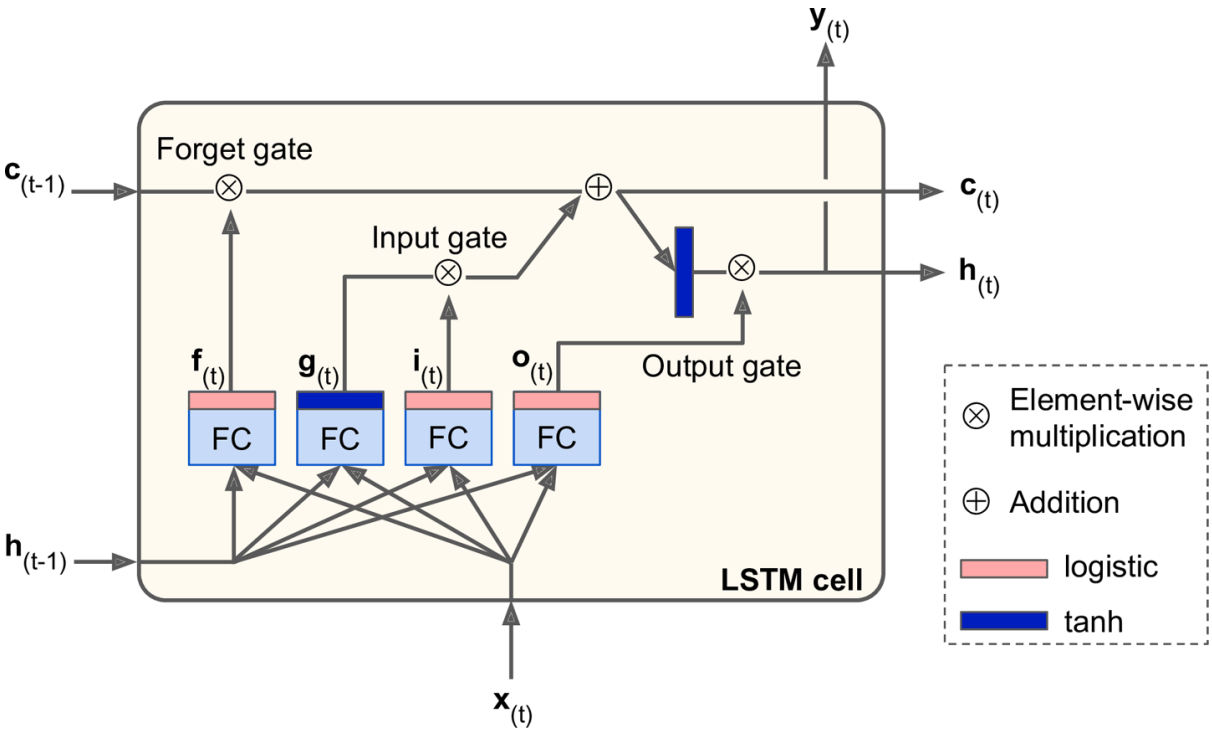
Figure 16: Predicting the evolution using the trained ELM for a random parameter value.



4.4 Using Recurrent Networks (LSTM) for Reconstruction

Initial attempts using a single Reservoir Computing (RC) model (like Echo State Network - ESN) trained on combined time-series data from different parameter values (p) failed to accurately reconstruct the bifurcation diagram, often yielding noise-like output 12. This occurred because the fixed reservoir and single readout could not effectively disentangle the parameter-dependent behaviour required for reconstruction.

Figure 17: Long Short-Term Memory (LSTM) architecture



An alternative approach using a Long Short-Term Memory (LSTM) network, a type of Recurrent Neural Network (RNN), proved more successful. The key differences enabling this approach to work better are:

1. **Parameter as Explicit Input:** Unlike the previous ESN implementation 4.3, the LSTM model is explicitly designed to take both the current state x_t and the bifurcation parameter p as inputs to predict the next state x_{t+1}. The learned mapping is:

$$\hat{x}_{t+1} = g_{\text{LSTM}}(x_t, p; \theta) \tag{3}$$

where θ represents all trainable weights (including recurrent connections) of the LSTM.

2. **Learning Conditional Dynamics:** The single LSTM model is trained using data aggregated from all available training parameters $\{p_n\}$. By receiving p as input, the network learns the conditional dynamics $x_{t+1}|(x_t, p)$ across the sampled parameter range, rather than averaging the dynamics as in the initial ESN attempt.
3. **End-to-End Training:** All weights θ within the LSTM are adjusted during training (via backpropagation). This allows the network's internal recurrent dynamics to adapt specifically for modelling how the system's behaviour changes with p, offering potentially more representational power than the fixed, random reservoir of a standard ESN/ELM.
4. **Reconstruction via Iteration:** The trained LSTM g_{LSTM} is iterated autonomously for any desired parameter p^* to generate attractor points:

$$\hat{x}_{t+1} = g_{\text{LSTM}}(\hat{x}_t, p^*; \theta) \tag{4}$$

In short, the LSTM's ability to directly incorporate the parameter p as input and learn the parameter-conditioned dynamics through full network training seems to be a good explanation of its superior performance compared to the initial single-ESN implementation, which lacked this explicit parameter handling. This LSTM approach remains distinct from the ELM + PCA method 4.3, which identifies parameter space through PCA on the output weights of separately trained predictors.

Figure 18: Comparison shown with reconstruction of the Bifurcation diagram using LSTM. In the reconstructed BD graph, the highlighted green region is the only region for which the model was trained. The rest is reconstructing how the model thinks the system would evolve for different parameter values.

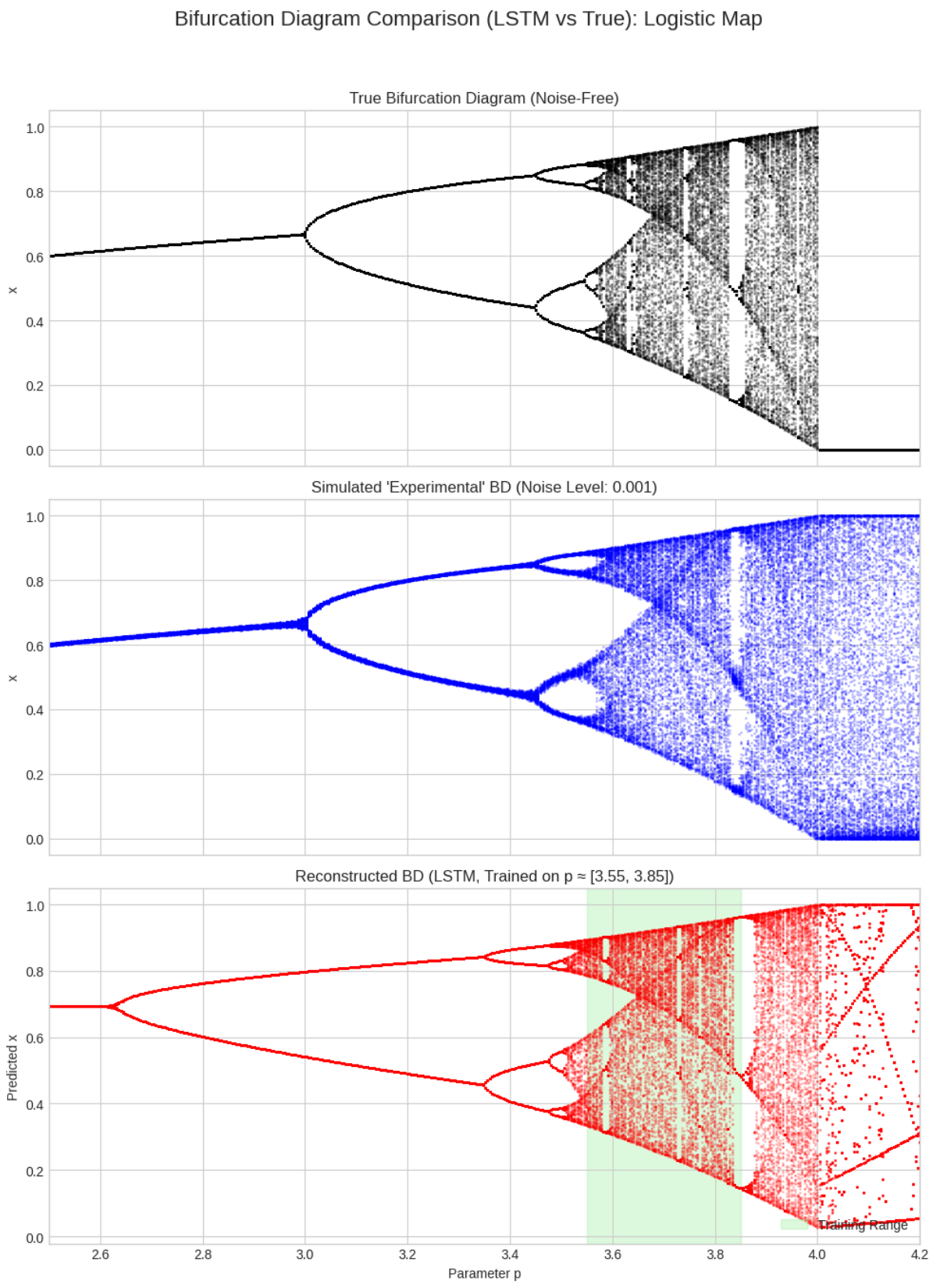


Figure 19: Reconstruction of evolution of $x_{(t+1)}$ vs x_t for different parameter values.

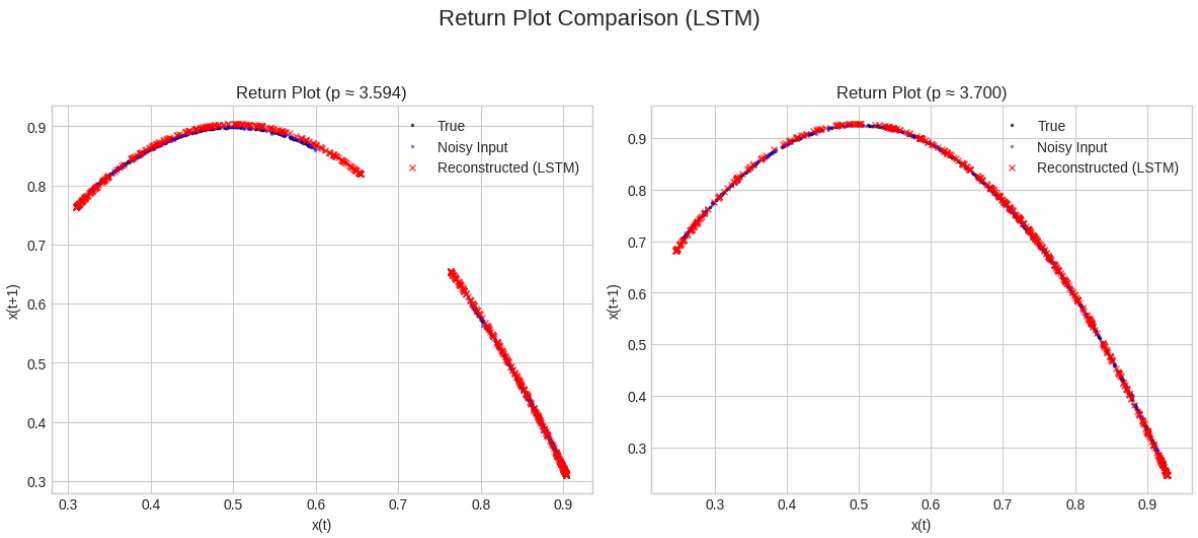
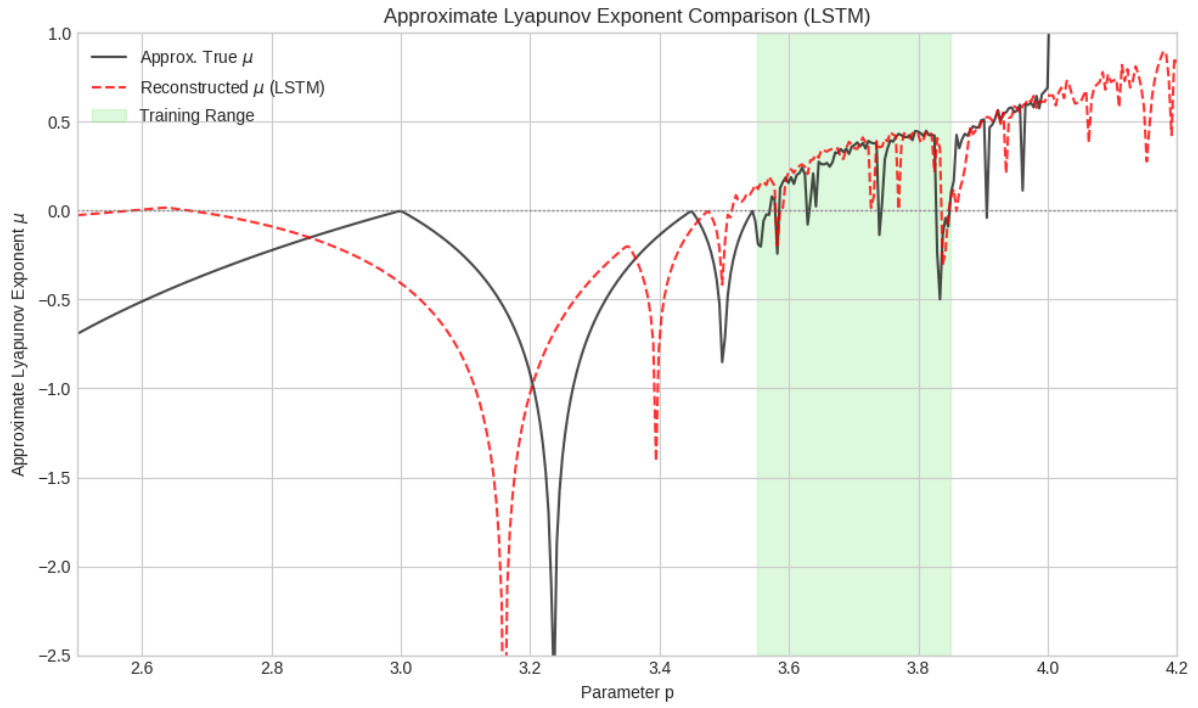


Figure 20: Using the model to estimate the approximate Lyapunov exponent for different parameter values. Again, like others, the green region highlights the training data limit.



The primary difference in terms of usability was that the LSTM model took around **46 seconds** to train while the regression ELM took **less than a second**.

This performance gain is especially because of training only the output layer is where reservoir computing performs extremely well. But due to compute being so much cheaper and efficient in most modern problems, sequential models - be it as rudimentary as RNNs, or newer approaches like Transformers or State Space Models (Mamba) Gu and Dao (2024) could be a better fit for such problems.

5 Conclusion

This project explored the application of reservoir computing (RC), one of the tasks that involves utilising an Extreme Learning Machine (ELM) combined with Principal Component Analysis (PCA), to reconstruct bifurcation diagrams from simulated noisy time-series data. The goal was to emulate data analysis from physical systems like electronic circuits, using the logistic map as a case study. While the approach successfully reproduced the broad features of the period-doubling route to chaos present in the logistic map, it fundamentally failed to capture fine structures within the chaotic regions or to accurately extrapolate beyond the parameter range used during training. These limitations underscore the challenges posed by noise and the sensitivity to RC hyperparameter configurations.

Beyond the specific results, this investigation provided valuable insights into the practical implementation and characteristics of RC. I learned about the core principles of RC, including its massive efficiency while training when compared to traditional recurrent networks, and its potential for modelling nonlinear systems directly from time-series data. The project highlighted scenarios where RC's ability to process temporal information with minimal training overhead could be advantageous, particularly in contexts involving physical systems where obtaining clean data or precise system models is difficult.

However, the difficulties encountered in accurately reconstructing complex dynamics and extrapolating outside the trained parameter space emphasise the trade-offs inherent in the RC approach and the need for tuning and potentially hybrid methods for more demanding prediction tasks.

In hindsight, I feel Recurrent networks are more suited for all of these tasks, especially as now computing is not a major limiting factor in most research work.

6 Future Work

Future efforts could focus on improving the fidelity and robustness of the bifurcation diagram reconstruction. Optimisation of RC hyperparameters (reservoir size, spectral radius, leaking rate, input scaling, regularisation) as well as using techniques like Bayesian optimisation could be essential to potentially overcome the observed limitations in capturing fine structures and improving extrapolation.

Applying the methodology to a wider array of dynamical systems would further test its generality. This includes investigating higher-dimensional maps, continuous-time systems (requiring considerations for numerical integration or time-delay embedding), systems exhibiting different bifurcation types (Hopf, saddle-node), and coupled systems to assess scalability.

Comparing the performance with alternative RC architectures, such as deep reservoir networks or different readout mechanisms beyond ELM, could identify more effective configurations for this specific task.

Finally, validating the simulation results by applying the reconstruction technique to time-series data obtained from actual physical hardware, such as electronic circuits, pendulums, or other systems, which kind of motivated this study, would provide insights into its real-world applicability and limitations.

References

R. Arun, M. S. Aravindh, A. Venkatesan, and M. Lakshmanan. Reservoir computing with logistic map. *arXiv preprint arXiv:2401.09501*, 2024.

Wikimedia Commons. File:freqgenschema.png — wikimedia commons, the free media repository, 2024. URL `https://commons.wikimedia.org/w/index.php?title=File:FreqGenSchema.png&oldid=955422627`. [Online; accessed 26-April-2025].

Matteo Cucchi, Steven Abreu, Giuseppe Ciccone, Daniel Brunner, and Hans Kleemann. Hands-on reservoir computing: a tutorial for practical implementation. *Neuromorphic Computing and Engineering*, 2, 08 2022. doi: 10.1088/2634-4386/ac7db7.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL `https://arxiv.org/abs/2312.00752`.

Y. Itoh, S. Uenohara, M. Adachi, T. Morie, and K. Aihara. Reconstructing bifurcation diagrams only from time-series data generated by electronic circuits in discrete-time dynamical systems. *Chaos*, 30(1):013128, 2020. doi: 10.1063/1.5119187.

Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 01 2001.

Wolfgang Maass. Liquid state machines: Motivation, theory, and applications. 01 2010. doi: 10.1142/9781848162778_0008.

Swarnendu Mandal, Sudeshna Sinha, and Manish Dev Shrimali. Machine-learning potential of a single pendulum. *Physical Review E*, 105:054203, 2022. doi: 10.1103/PhysRevE.105.054203.

Junhyuk Woo, Soon Ho Kim, Hyeongmo Kim, and Kyungreem Han. Characterization of the neuronal and network dynamics of liquid state machines. *Physica A: Statistical Mechanics and its Applications*, 633:129334, 2024. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2023.129334. URL `https://www.sciencedirect.com/science/article/pii/S0378437123008890`.

Yuanzhao Zhang and Sean Cornelius. Catch-22s of reservoir computing. *Physical Review Research*, 5, 09 2023. doi: 10.1103/PhysRevResearch.5.033213.

Acknowledgement

I would like to thank **Prof. Gaurav Dar** for his guidance, insights and motivating me throughout this project.